

REMARKS

In an office action mailed January 14, 2005 (paper no. 20050103), the specification was objected to for noted informalities. Claim 1 was objected to for noted informalities. Claims 1-7 and 15-20 were rejected under 35 U.S.C. 112 for reasons that are not apparently clear. Claims 1-20 were variously rejected under 35 U.S.C. 102 and 103 as being anticipated by or unpatentable over Chiang. The objections and rejections are respectfully traversed.

Objections to the Specification

Appropriate changes have been made.

Objections to Claim 1

Appropriate changes have been made.

Rejections under 35 U.S.C. 112.

Claim 1 stands rejected under 35 U.S.C. 112 as allegedly being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. In particular, the Examiner asserts that it is not clear whether the word operator is referring to a user or a mathematical operator. Nowhere in the specification is any mention made of mathematical operators performing any function, whereas numerous references are made to users and developers. The term "operator" was selected to encompass any actor described in the specification, such as programmers, users, developers, or other such terms. If the Examiner remains unclear on the term "operator," the Applicants will amend the specification to replace "user," "programmer," "developer" or other similar terms with the terms "operator or user," "operator or programmer," "operator or developer," or other suitable terms. However, the Applicants believe that such a change is clearly unnecessary, and that the rejection of claim 1 under 35 USC 112 should therefore be withdrawn.

Claim 15 stands rejected under 35 U.S.C. 112 as allegedly being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. In particular, the Examiner asserts that it is not clear whether "one or more classes"

also refers to "developer classes." However, the claim itself makes it clear that there is a difference. The claim includes "a primary code generator receiving one or more user selections from one or more classes and generating primary software code," and "a developer code generator receiving the primary software code and one or more user selections from one or more developer classes and generating developer software code." If the "one or more developer classes" are the "one or more classes," then the claim would have to be amended to read "a developer code generator receiving the primary software code and the one or more user selections from the one or more ~~developer~~ classes and generating developer software code," or perhaps "a developer code generator receiving the primary software code and one or more additional user selections from the one or more ~~developer~~ classes and generating developer software code" to provide proper antecedent basis, where as if the terms are different, then no modifications are needed. A construction of a claim term that creates problems with the antecedent basis of claim terms should not be asserted by the Examiner over a construction that does not create such problems. The rejection of claim 15 under 35 USC 112 should therefore be withdrawn.

As there is no indefiniteness under 35 USC 112 with claims 1 and 15, the rejection of the dependent claims under 35 USC 112 should also be withdrawn.

Rejections under 35 U.S.C. 102 and 103

Claims 1-20 were variously rejected under 35 U.S.C. 102 and 103 as being anticipated by or unpatentable over Chiang. The rejections are respectfully traversed.

Chiang fails to provide a basis for the rejection of claims 1-20 under 35 U.S.C. 102 and 103. In regards to claim 1, the Examiner asserts that Chiang discloses a "handler class system generating handler class code, wherein the handler class code includes one or more states for each user interface feature of the user interface class," citing to event handler 415. However, event handler 415 of Chiang "coordinates between the application framework 410 and the application business logic 420," paragraph 0036. If business logic code is not available, it is generated by web application generator 205, paragraph 54. Web application generator 205 is not an automatic system but rather a system that combines human activities of graphic designers/business analysts 210 and web developers 215. In contrast, claim 1 includes a "user interface class system generating user interface class code, wherein the user interface class code includes two or more user interface features that can be

assembled into a user interface by an operator; and a handler class system generating handler class code, wherein the handler class code includes one or more states for each user interface feature of the user interface class; wherein the user interface class and the handler class cause the selected user interface features and associated states for the user interface features to be generated when the user interface code is executed." Since the user interface features of claim 1 each have associated handler classes, there is no need to have programmers perform any manual functions. Put another way, the system of Chiang allows a user to develop a user interface, and if there happens to be some underlying functional code that "fits" the user interface, then it is matched up, otherwise, new code is manually programmed. In contrast, claim 1 eliminates manual programming by allowing designers to select user interface features that have pre-existing states in handler class code. This is not a case where claim 1 merely covers automation of manual actions – instead, users select from a predetermined set of interface features that have associated handler codes, and the selected user interface features and associated states for the user interface features are generated when the user interface code is executed. In order for these functions to occur, all possible interface features and associated states must be identified and coordinated with each other. Chiang clearly fails to anticipate claims 1, as if all possible interface features and associated states were identified, then there would be no need for the web developers 215 of Chiang.

Claim 2 includes the system of claim 1 wherein the user interface class system further comprises a developer user interface class system generating a developer user interface class that operates in conjunction with the user interface code to provide modified user interface features. The Examiner alleges that graphic designers/business analysts 210 and graphical user interfaces 405 disclose this element, but the specification of Chiang makes it clear that this contention is incorrect. Graphic designers/business analysts 210 of Chiang are people performing manual activities, and graphical user interfaces 405 are the results of those manual activities. The developer user interface class system is not a person but rather a system that operates in conjunction with the user interface code to provide modified user interface features. For example, if a software package includes user interfaces and a developer wants to modify user interface features, the developer user interface class system can be used to do so. The developer user interface class created using the developer user interface class system does not require the user interface code to be modified, such that if the developer user interface class does not provide the correct functionality, it can be discarded and a new developer user interface class can be created, as opposed to modifying the user interface code.

Claim 3 includes the "system of claim 1 wherein the user interface class system further comprises a developer handler class system generating a developer handler class that operates in conjunction with the user interface code to provide modified user interface states." The Examiner has identified the identical structure for the handler class system and the developer handler class system, which is a clear indication that the Examiner's construction of these terms is incorrect. Not only is that construction contrary to the definitions in the specification, it is also contrary to the plain meaning, which requires items that have two different names to be two different things. Furthermore, that construction is contrary to Chiang, which shows that web developers 215 provide input to the web application generator 205, which the Examiner asserts generates the user interface code. However, in regards to claim 3, the Examiner apparently asserts that web application generator 205 doesn't generate the user interface code but instead operates in conjunction with the user interface code to provide modified user interface states. This makes no sense, unless the Examiner is merely construing the user interface states to be the same as the modified user interface states in order to avoid having to find a reference that shows both. In any event, the developer handler class system must be different from the handler class system – the Examiner cannot arbitrarily equate these two systems as being identical simply to reject the claims over a reference that only shows a single system (and a system comprised of manual programmers, at that).

In regards to claims 4 and 5, which include a site-specific user interface class system and a site-specific handler class system, respectively, the Examiner claims that it is "obvious to one of ordinary skill in the art at the time the invention was made to reiterate the process of modification." This, perhaps, explains the flaw in the Examiner's logic. The (1) user interface class system, developer user interface class system and site-specific user interface class system (2) handler class system, developer handler class system and site-specific handler class system are not merely iterative code modification processes that are manually performed by programmers, but rather independent systems that operate at different locations. The Examiner has improperly construed the system claims to be method claims – as described in greater detail below in regards to other claims, this construction is contrary to the construction of these terms that must be afforded in light of the specification and other claims.

Claims 6 and 7 identify various user interface and handler classes, which are enabled by the specification and which describe user interface classes and handler classes that allow user interfaces to be generated and modified by developers and for site-specific applications. As described in the

specification, these user interfaces and handler classes allow changes to be readily made by developers and at site-specific locations without the need for iterative modification of code, by creating developer and site-specific user interface classes and handler classes that are fully compliant with the original user interface classes and handler classes and with each other. In this manner, changes can be quickly rolled back by substituting new developer and site-specific user interface classes and handler classes, and without the need for cumbersome, time-consuming, and error prone iterative modification processes. The Examiner's rejection of these claims is also defective as a matter of law because Chiang entirely fails to describe all of the user interfaces and handler classes of claims 6 and 7, respectively. The following terms are missing entirely from Chiang – general ledger, accounts receivable, accounts payable, purchase order, and shipping order. How can Chiang provide a basis for rejection of these claims under 35 U.S.C. 102 when so many important terms are completely missing from Chiang? These are not mere obvious variations – the Applicants call the Examiner's attention to pages 15 through 30 of the specification where the interrelation of these and other user interface classes and handler classes are described. Absolutely none of the prior art cited by the Examiner in any way addresses such functionality.

Claim 8 includes a "method for generating user interface code comprising: receiving a selection of a user interface feature from a user interface class; retrieving a handler associated with the user interface feature that includes one or more states; and generating one or more code elements that cause a user display to be generated when executed that includes the user interface feature having the one or more states." The Examiner's improper construction of the claim terms is again evident, as the Examiner cites to sections of Chiang that describe manual processes being performed by programmers to iteratively modify code. No manual processes are required by claim 8 after the user interface feature is selected.

Claim 9 includes the "method of claim 8 further comprising: receiving a selection of a developer user interface feature from a developer user interface class; and generating one or more code elements that cause the user display to be generated that includes the developer user interface feature." Claim 10 includes the "method of claim 8 further comprising: receiving a selection of a developer user interface state from a developer handler class; and generating one or more code elements that cause the user display to be generated that includes the developer user interface state."

As previously discussed, this not a manual process of iterative modification, but rather the selection of developer user interface features and states from developer user interface and handler classes that

cooperate with, and not modify, the existing user interface features and handlers. Unlike the cumbersome prior art teachings of Chiang, where code is continuously modified and where versions must be rolled back in entirety to correct a mistake, the present invention allows individual user interfaces and handlers to be added that supplement (and that do not modify) existing user interfaces and handlers, so that mistakes can be readily corrected and modifications can be readily made.

Claims 11 and 12 add the site-specific user interface features and handlers, which, as described, are not merely iterative modifications of code that change it and render it unusable for other site-specific applications. Instead, the method of claims 11 and 12 allow users to replace a set of site specific user interface features and/or handlers with a new set, automatically modifying the user interface and functionality without the need to modify code from scratch. The rejection of these claims for the reasons stated by the Examiner are improper and should be withdrawn.

The rejection of claims 13 and 14 is improper for the reasons previously discussed in regards to claims 6 and 7.

The rejection of claim 15 is likewise improper, as Chiang entirely fails to disclose a primary code generator receiving one or more user selections from one or more classes and generating primary software code. An application directory is not a class, examples of which are provided in the specification and which will not be recounted here. The Examiner relies on inherency to supply the missing primary code generator element, but the inherency argument is flawed. The doctrine of inherency requires that the missing element must necessarily be present, but Chiang itself identifies what performs the function of the missing primary code generator element – web developers 215. However, these manual programmers write customer framework code for web pages after the web page design is completed, paragraphs 26-28 of Chiang. Claim 15 only requires that a user select one or more classes, after which the generation of code occurs without manual activity.

Likewise, Chiang fails to disclose a developer code generator receiving the primary software code and one or more user selections from one or more developer classes and generating developer software code. Incredibly, the Examiner cites to the same section that he relied on for support in rejecting the first element! The Examiner's construction thus eliminates an entire element of the claim, which is improper.

Likewise, the Examiner again relies on the same section of Chiang to reject the elements of "a primary code editor modifying one or more of the classes; and wherein the modifications made to the one or more of the classes by the primary code editor result in the generation of code that is

compatible with the developer software code." As previously described, and as clearly evident in claim 15, the modifications made to the one or more of the classes by the primary code editor result in the generation of code that is compatible with the developer software code. Thus, the process works both ways – primary code can be modified and still be compatible with the downstream modifications made by the developer systems and site specific systems, and modifications made by the developer systems and site specific systems are compatible with the primary code. The Examiner's improper construction equates modifications at any of the primary, developer, or site-specific stages with an iterative modification process on a single code set – as previously described, this is clearly wrong because once the code has been iteratively modified, it is impossible for the primary code to be modified and for those modifications to be compatible with a version of the code that has been iteratively modified. The modified primary code would need to be iteratively modified in the same manner as the original primary code – the invention avoids this time consuming and expensive process using a multi-tiered architecture where changes made at any tier are backwards and forwards compatible with other tiers.

It is also noted that the Examiner relies on inherency for both the editor and compatibility, but again is mistaken. As previously discussed, the standard for inherency is that the element must necessarily be present in the prior art. While a text editor may be necessarily present in Chiang to modify input files, claim 15 does not disclose a text editor but rather a primary code editor modifying one or more of the classes. Thus, it is not input files that are being modified, but classes, which are not even disclosed in Chiang. It is axiomatic that an element cannot be inherent in a reference when the function performed by that element and the item that the element functions on is not even disclosed in the reference. Likewise, compatibility is not only not inherent, but it is impossible under the system of Chiang. Claim 15 is addressing the downstream compatibility process – the Examiner is claiming that primary code, after being iteratively modified by a developer, would inherently be compatible with a new version of the primary code. That is simply not the case – it is obvious to one of ordinary skill in the art that when a program is modified by two different programmers in two different manners, that it would only be by sheer luck that the two resulting programs would be compatible. Compatibility in that scenario is not only not inherent, but disaster is almost guaranteed.

The rejections of claims 16 through 20 are likewise improper. Withdrawal of the rejections and allowance of all pending claims is respectfully requested.

CONCLUSION

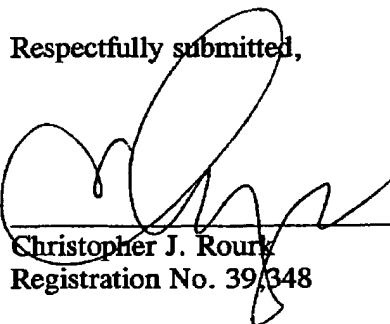
In view of the foregoing remarks and for various other reasons readily apparent, Applicants submit that all of the claims now present are allowable, and a Notice of Allowance is courteously solicited.

If any impediment to the allowance of the claims remains after consideration of this amendment, a telephone interview with the Examiner is hereby requested by the undersigned at (214) 939-8657 so that such issues may be resolved as expeditiously as possible.

No fee is believed to be due at this time. If any applicable fee or refund has been overlooked, the Commissioner is hereby authorized to charge any fee or credit any refund to the deposit account of Godwin Gruber, LLP, No. 50-0530.

Respectfully submitted,

By:



Christopher J. Rourke
Registration No. 39,348

GODWIN GRUBER LLP
1201 Elm Street, Suite 1700
Dallas, Texas 75270
(214) 939-8657 (Telephone)
(214) 760-7332 (Facsimile)
crourke@godwingruber.com (email)